武汉大学计算机学院

课程实验(设计)报告

课程名称： 基于WinCE 6.0的“街头霸王”游戏设计与实现

专业、班级： 计科<1>班

组 长： 李坚松 2012301500026

组 员： 李 标 2012301500028

组 员： 童立成 2012301500029

2015/12/18

目录

1. 人员分工

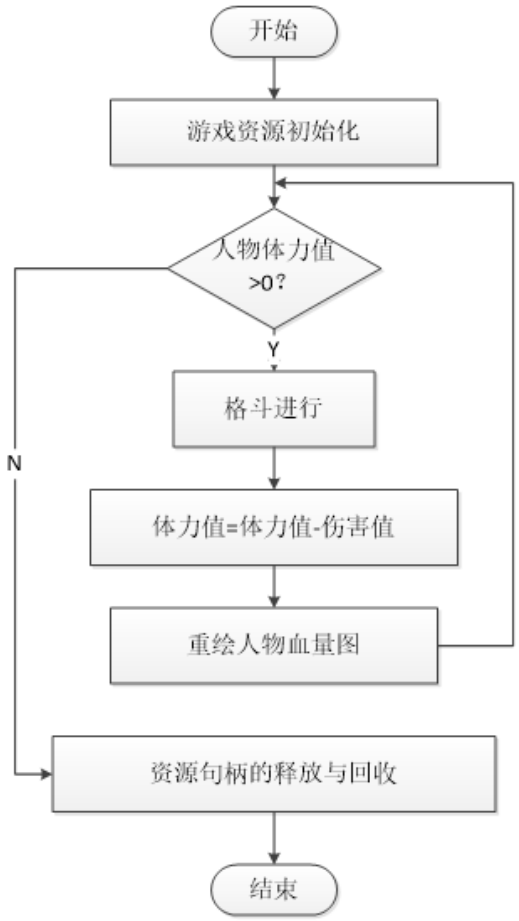
姓名	身份	分工
李坚松	负责人	游戏设计，模型制作，整个游戏代码的实现，实验报告的书写
李标	组员	游戏设计，模型制作，代码调试，测试
童立成	组员	游戏设计，模型制作，测试

2. 选题介绍

此次的选题为基于 WinCE 6.0 操作系统下“街头霸王”游戏的设计与实现，之所以在嵌入式的课程设计上选做游戏，一方面是因为游戏开发本身就很有意思，通过游戏开发可以很轻松地掌握一门编程语言，另一方面是因为嵌入式设备本身拥有的 API 就很适合游戏开发，能在嵌入式平台上操纵一款游戏运行还是很有意思的一件事。这款游戏是由日本 CAPCOM 公司于 1987 年首次推出的格斗类单机游戏系列，截止目前，无论是游戏的剧情，还是人物的建模都已经很成熟了，所以在短时间内基于这些已有的资源开发出一个 Demo 版本还是可行的。

此外，WinCE 操作系统本身就已经封装了大量的 API，利用这些 API 就可以实现游戏开发的核心技术：**游戏缓冲技术、GDI 映射、透明遮罩、游戏动画技术、键盘消息处理、粒子系统、碰撞检测**等。比如说，利用 WinCE 提供的 GDI 接口，像 LoadBitmap、StretchBlt、TransparentBlt、BitBlt 等，这些函数可以很方便地将设计好的人物模型加载到游戏场景中。而且，WinCE 提供的 SDK 中也封装了定时器、Windows 的消息映射机制，利用这些 API 很容易就可以实现游戏的动画效果和响应键盘消息。鉴于开篇部分篇幅有限，上述几个技术的详细实现原理会在“游戏模型设计与实现”的模块中给出详细的介绍。

由于课程实验时间有限，所以此次的游戏设计是在原版的《街头霸王》游戏剧情的基础上进行一定的删减实现的。此次的课程实验主要实现的就是利用外接的键盘设备，控制玩家与电脑端的人物进行对打，直到其中一方的体力值(游戏人物的血量)消耗殆尽，游戏结束。整个程序的思路也很清晰，如下面的流程图所示：



在上图中，游戏资源就是自己事先设计好的人物模型，以 bmp 位图的文件格式存储，在初始化的同时会将双方的体力值，即血量值设定成一个固定的数值。在双方进行格斗时，会根据游戏人物打出招式的不同，来给对方造成不同的伤害值。被攻击方的体力值会在原有体力值基础上再减去攻击方打出的伤害值。为了加强游戏的效果，每个游戏人物的血量图也会根据游戏人物的体力值进行实时地更新。游戏结束以后，相应的资源句柄自然要进行释放与回收。上述流程图的每个部分都对应着一个函数，后面的章节会给出详细的说明。

整个游戏主要场景设计如下图所示。

由于玩家的打法控制比较灵活，电脑端自然也就需要根据玩家的每个动作进行实时地判

断与分析，这部分属于游戏 AI 的范畴，本游戏在这部分也有所实现。此外，为了增强游戏场景的真实感，游戏中也添加了人物的表情动作、烈火燃烧的场景以及雪花粒子，使游戏场景更显逼真。

3. 开发环境

主要的开发环境如下表：

编程环境：	VS2005 +NewQoMobile SDK
操作系统：	WinCE 6.0
开发板：	OK6410
制造商：	Forlinx

下面对这些开发环境做出简单的介绍。

3.1 WinCE 6.0 简介

Windows Embedded Compact，即 Windows CE，简称 WinCE，它是微软公司嵌入式、移动计算平台的基础。作为一个开放的、可升级的 32 位嵌入式操作系统，它是基于掌上型电脑类的电子设备操作系统。作为精简版的 Windows 95，WinCE 操作系统是 Windows 家族中的成员，它也是专门为设计给掌上电脑（HPCs）以及嵌入式设备所使用的系统环境。Windows CE 被设计成针对小型设备通用操作系统，利用它就可以设计一层位于内核和硬件之间代码来设定硬件平台，这就是所谓的硬件抽象层（HAL）。自从 1996 年微软推出 Windows CE 1.0 以来，Windows CE 一共经历了 7 个不同的版本。此次选用的是应用比较广泛的 6.0 版本。

3.2 Forlinx 简介

飞凌嵌入式，简称 Forlinx，是一家专注于嵌入式开发的公司，其产品主要面向中低端的嵌入式领域。它的产品也是基于 ARM 平台的，先后推出了 S3C2440 开发板，9263 开发板和 S3C6410 开发板等多个系列的 ARM 平台。此次的实验设计使用的 OK6410 开发板也是该公司旗下的一个子产品。

3.3 OK6410 开发板简介

整个开发板的实物图如下图所示：

这个开发板主要适用于高端消费类电子产品、工业控制、车载导航、多媒体终端、电子付费终端、行业 PDA 等领域。OK6410 采用通用 0.8mm 间距镀金连接器，其液晶屏置于核心板上方，并且液晶屏与底板用螺丝固定，可以进行拆装。对于 RAM，它提供 128M 和 256M 两个版本，用户可根据需要自由选择。下面给出它主要的硬件参数介绍。

3.3.1 核心板资源

Samsung S3C6410 处理器，ARM1176JZF-S 内核，主频 533MHz/667MHz
长宽尺寸仅 5CM*6CM
引脚 320 个
128M 字节 DDR 内存，256M Byte Nand Flash，带有 8Bit 硬件纠错算法
256M 字节 DDR 内存，SLC 1G Byte Nand Flash，10 万次写入（10 倍于 MLC）
12MHz、48MHz、27MHz、32.768KHz 时钟源
采用进口高质量板对板接插件，确保长期运行可靠性
支持 5V 电压供电

3.3.2 底板资源

1 个复位按键，采用专用芯片进行复位
采用 8 位拨码开关设置系统启动方式
共 4 个串口，包括 1 个五线 RS 232 电平串口（DB9 母座）和 3 个三线 TTL 电平串口（20pin 2.0mm 间距插头座）
1 个 100M 网口，采用 DM9000AE，带连接和传输指示灯
1 个 USB HOST 插口，支持 USB1.1 协议，可插鼠标、U 盘等
1 个 USB Slave 接口，支持 USB2.0 协议，使用 Mini-USB 插座，可与 PC 连接
1 个高速 SD 卡座。可以实现 SD Memory 功能和 SDIO 功能
1 个无线网卡接口（WIFI）
3 个 3.5mm 标准立体声音频插座。其中包括 1 个音频输出插座，可与耳机连接；1 个话筒输

入插座；1 个线路输入插座
触摸板接口支持 4 线电阻式触摸板
1 路 CVBS 输出接口（PAL / NTSC）
1 个 CMOS 摄像头接口，支持 ITU-RBT601/656 8 位模式，使用 10*2 插针连接器
内部实时时钟，带有后备锂电池座，断电后系统时间不丢失
1 个 JTAG 接口，使用 10*2 插针连接器
1 个红外模块
4 个 LED
2 路 IIC
1 个蜂鸣器
3 个 10×2 插针扩展口。一个 包括 1 路 GND、1 路 DA、8 路 AD、10 路 IO、1 路 SPI。 一个用来扩展 8×8 矩阵键盘，另一个可连接 3 个 TTL 电平的串口和 6 路 IO 口，这 3 个串口中，包括 1 个五线串口和 2 个三线串口

4. 游戏设计核心技术

上面主要介绍了选题背景和开发环境，下面开始进入正题。此次“街头霸王”游戏的核心技术主要有：透明遮罩、GDI 映射方式、三缓冲技术、游戏动画技术、键盘消息处理等。因为 WinCE 操作系统的 API 与 Windows 的 API 大体相仿。在介绍这些技术之前先介绍一下几个核心的 Windows API 函数。

4.1 核心 API

4.1.1 WinMain

如同 main 函数是 dos 程序的入口点，WinMain 函数是 Windows 应用程序的入口点。其函数原型如下：

```
int WINAPI WinMain( __in HINSTANCE hInstance,
                    __in_opt HINSTANCE hPrevInstance,
                    __in_opt LPWSTR lpCmdLine,
                    __in int nShowCmd );
```

这里的__in、__in_opt 其实就是一个宏的标识，第一个参数 hInstance 标识该程序当前运行的实例句柄。当一个程序在 Windows 下运行时，它唯一对应一个运行中的实例，也只有运行中的实例程序才有资格分配到实例句柄。一个应用程序可以运行多个实例，系统都会给该实例分配一个句柄值，并且通过 hInstance 参数传递给入口点 WinMain 函数。第二个参数 hPrevInstance 表示当前实例的前一个实例的句柄，在 Win32 程序中，包括 WinCE，hPrevInstance 为 0 并且可以忽略。第三个参数 lpCmdLine 指定了传递给应用程序的命令行参数，注意，在 Windows CE 下的命令行参数字符串都是 Unicode 字符串，所以是 **LPWSTR** 类型的，与桌面版的 Windows 略有不同，在桌面版的 Windows 中，这个字符串是 ASCII 字符串，是 **LPSTR** 类型的。最后一个参数 nShowCmd 指定了程序主窗口的初始状态，可以设

置窗口最大化、最小化还是隐藏等。

4.1.2 MSG 结构体

Windows 程序中的所有消息都是用 MSG 结构体来表示的。其原型如下：

```
typedef struct tagMSG {
HWND hwnd;
UINT message;
WPARAM wParam;
LPARAM lParam;
DWORD time;
POINT pt;
} MSG;
```

其中，hwnd 指定了消息所处的窗口，message 指定了消息的标识符，wParam 和 lParam 指定了消息的附加信息，time 表示投递到消息队列中的时间，pt 表示投递到消息队列中时鼠标的当前位置。

一个完整的窗口创建需要经历四个步骤，窗口类的设计，窗口类的注册，窗口的正式创建，窗口的显示与更新等。其中，窗口类的设计步骤，设置窗口的特征，如光标、图标、背景颜色等属性信息。下面这一段代码就实现了窗口类的设计：

```
WNDCLASS ws;
ws.cbClsExtra    = 0;
ws.cbWndExtra    = 0;
ws.hbrBackground = (HBRUSH)GetStockObject(GRAY_BRUSH);
ws.hCursor       = NULL;
ws.hIcon         = NULL;
ws.hInstance     = hInstance;
ws.lpfnWndProc = WndProc;
ws.lpszClassName = TEXT("Hello");
ws.lpszMenuName   = NULL;
ws.style         = CS_VREDRAW | CS_HREDRAW;
```

窗口设计完之后需要注册，下面的代码就实现了窗口的注册：

```
RegisterClass(&ws);
```

一般情况下，在 WinMain 结束之前，最好调用 UnRegisterClass 把创建的窗口类注销，这样做可以使程序更加稳定，当然不调用也没什么影响。

注册完之后就可以正式创建窗口，调用 CreateWindow 就可以实现窗口的正式创建：

```
UINT sysWidth=GetSystemMetrics(SM_CXSCREEN);
UINT sysHeight=GetSystemMetrics(SM_CYSCREEN);
HWND hwnd = CreateWindow(TEXT("Hello"),TEXT("Street Fighter V1.0"),WS_VISIBLE |
  WS_BORDER | WS_SYSMENU /*| WS_MINIMIZEBOX*/ /*| WS_MAXIMIZEBOX*/ | WS_CAPTION,
      0,0,sysWidth,sysHeight,
      NULL,NULL,hInstance,NULL);
```

然后调用 UpdateWindow 和 ShowWindow 就可以实现窗口的显示与更新：

```
UpdateWindow(hwnd);
ShowWindow(hwnd,nShowCmd);
```

经过上面四个步骤以后，还需要写一个消息循环，不断地从消息队列中取出消息，并执行相应的响应函数。Windows CE 下的消息响应体系主要有两种，一种是以 GetMessage 为核心的消息循环体系，另一种是以 PeekMessage 为核心的消息循环体系。

4.1.3 窗口过程函数

一个窗口类的窗口行为由窗口过程的具体代码决定，窗口过程是窗口类定义的重要组成部分。发送到窗口的所有通知和请求都是由窗口过程处理的。这些通知都是以消息的形式传递。窗口过程函数的一般声明如下，名字可以不唯一，但是参数是固定的：

```
LRESULT CALLBACK WndProc(
HWND hwnd,
UINT message,
WPARAM wParam,
LPARAM lParam);
```

其中，第一个参数 hwnd 表示需要处理消息的那个窗口的句柄。第二个参数 message 表示待处理消息的 ID，第三、第四个参数表示消息的附加信息。一般都会写成如下形式：

```
LRESULT CALLBACK WndProc(HWND hwnd,UINT message,WPARAM wParam,LPARAM lParam)
{
    switch (message)
    {
    case WM_KEYDOWN:
        //具体处理过程
        break;
    default:
        return DefWindowProc(hwnd,message,wParam,lParam);
    }
    return 0;
}
```

4.1.4 GDI 函数

GDI 是图形设备接口，利用 GDI 可以绘制最基本的图形，比如说直线、曲线、位图、文字等。GDI 的函数很多，在这个游戏的开发中，人物的设计最终都是以 bmp 文件格式来存储的，每一种资源都对应着位图句柄，HBITMAP 类型的。位图的绘制主要有四个步骤，首先是调用 LoadBitmap 加载位图，然后调用 CreateCompatibleDC 建立兼容的 DC，因为要想对这个位图资源进行操作，首先需要有存放这个位图对象的地方，所以需要创建一个和设备环境兼容的内存设备环境。调用 SelectObject 就可以将相应的位图资源选入到内存设备环境中。最后就是贴图，这个处理很关键，因为游戏最终的显示效果就与贴图的方式有关。贴图的方式有很多种，主要有以下函数：BitBlt、StretchBlt、TransparentBlt 等。

函数 BitBlt，可以实现从源矩形中复制一个位图到目标矩形，必要时还可以按照目标设备设置的模式进行图像的拉伸或者压缩。其函数原型如下：

```
BOOL BitBlt(
```

```
_In_   HDC hdcDest,
_In_   int nXDest,
_In_   int nYDest,
_In_   int nWidth,
_In_   int nHeight,
_In_   HDC hdcSrc,
_In_   int nXSrc,
_In_   int nYSrc,
_In_   DWORD dwRop);
```

其中，参数 hdcDest 指向目标设备环境的句柄，第二、第三个参数表示目标矩形区域左上角的点的逻辑坐标，第四、第五个参数表示贴到目的 DC 的宽度和高度，hdcSrc 表示源设备环境句柄，nXSrc 和 nYSrc 表示源矩形区域左上角的点的逻辑坐标，dwRop 指定光栅操作代码，即贴图的方式。这些代码将定义源矩形区域的颜色数据，如何与目标矩形区域的颜色数据组合以完成最后的颜色。常用的有：SRCAND，通过使用 AND（与）操作符来将源和目标矩形区域内的颜色合并；SRCCOPY，将源矩形区域直接拷贝到目标矩形区域；SRCPAINT，通过使用布尔型的 OR（或）操作符将源和目标矩形区域的颜色合并。通过指定不同的光栅操作代码可以绘制出很多特殊的效果出来。比如说最后的游戏结束文字的贴图就是利用指定不同的光栅操作代码来实现**透明遮罩**的效果。

函数 StretchBlt 与函数 BitBlt 类似，可以实现从源矩形中复制一个位图到目标矩形，它比 BitBlt 多了两个参数，这两个参数可以控制贴图的具体位置。其函数原型如下：

```
BOOL StretchBlt(
HDC hdcDest,
int nXOriginDest,
int nYOriginDest,
int nWidthDest,
int nHeightDest,
HDC hdcSrc,
int nXOriginSrc,
int nYOriginSrc,
int nWidthSrc,
int nHeightSrc,
DWORD dwRop);
```

函数 TransparentBlt，可以实现对指定的源设备环境中的矩形区域像素的颜色数据进行位块(bit_block)转换，并将结果置于目标设备环境。其函数原型如下：

```
BOOL TransparentBlt(
HDC hdcDest,
int nXOriginDest,
int nYOriginDest,
int nWidthDest,
int hHeightDest,
HDC hdcSrc,
int nXOriginSrc,
int nYOriginSrc,
int nWidthSrc,
```

```
    int nHeightSrc,
    UINT crTransparent);
```

通过指定最后一个参数 crTransparent，可以把它作为源位图中的透明色，就可以把这种颜色过滤掉。比如说，在这个游戏中，游戏人物位图的背景色可以统一指定为白色 RGB(0,0,0)，通过下面的代码就可以实现将游戏人物与背景完美融合：

```
TransparentBlt(g_hdcMem,
        wndRect.right/20*19,wndRect.right/3,
        bmpFire.bmWidth/6,bmpFire.bmHeight,
        g_bufdc,
        (g_iHeroFileNum)*bmpFire.bmWidth/6,0,
        bmpFire.bmWidth/6,bmpFire.bmHeight,
        RGB(0,0,0));
```

到目前为止，所有的绘图函数都是相对于显示区域的左上角，以像素为单位进行绘图的。如果想将自己设计好的游戏模型贴到窗口中适当的位置，就需要知道当前窗口的位置坐标。在 Windows CE 中，提供了 GetClientRect 函数可以获取窗口客户区的大小。该函数的原型如下：

```
BOOL GetClientRect(
HWND hWnd, //  窗口句柄
LPRECT lpRect //  客户区坐标
);
```

如果函数成功，返回一个非零值。如果函数失败，返回零。要得到更多的错误信息，使用 GetLastError 函数就可以查看详细的错误信息。

4.2 缓冲显示技术

在编写游戏程序时，需要窗口每秒钟刷新多少次时，经常会遇到画面闪烁的问题。为了避免画面的闪烁，在游戏的画面显示中，这种技术称为双缓冲(double buffer)。双缓冲的思想比较简单，就是在其他地方再开辟一个存储空间，把所有的动画都渲染到这个地方，等渲染完成以后，再将画面整体拷贝显示到屏幕上。在 GDI 编程中，双缓冲的具体实现就是直接针对屏幕的窗口 DC，再创建一个与之兼容的内存 DC，把所有的资源都加载到这个内存 DC 中，在内存中对这些资源处理完以后就可以显示到窗口 DC 中。

双缓冲技术有时也被称为页面切换技术(page flipping)，为了使画面更加平滑，在双缓冲不能满足要求时，就可以使用三个缓冲区，这就是**三缓冲技术**(triple buffering)。，缓冲区的数量可以根据需要任意增加缩减，直接调用 CreateCompatibleDC 函数就可以创建一个与指定的 DC 兼容的内存 DC。

在我们的游戏设计中，游戏人物的绘制采用了三缓冲技术。也就是说新建立了与 g_hdc 兼容的两个 DC，一个 DCg_bufdc，另一个是 g_hdcMem。首先把所有的资源统一加载到 g_bufdc，再进行透明化处理等渲染操作，将渲染处理完的结果从 g_bufdc 贴图到 g_hdcMem(注：g_hdcMem 可以在初始化的时候贴上一张与窗口大小相同的空位图)，最后再将所有渲染处理完的画面整体拷贝显示到屏幕 g_hdc 中。这样就有效地避免了屏幕的闪烁问题。

4.3 游戏动画

在 2D 的游戏中，播放动画的方式一般有两种，一种是直接播放视频文件，另一种是利用连续贴图的方式来达到动画显示的效果。很明显，在这个游戏中，游戏人物的控制不能通过播放视频文件的方式来实现。因此只能采用第二种形式，对于第二种方式的实现，主流的技术主要有两种，一种是“定时器”技术，一种是“游戏循环”技术。

在 Windows 中，定时器是一种输入设备，它周期性地在指定一个时间间隔就以 WM_TIMER 消息的形式通知应用程序一次。通过调用 SetTimmer 函数可以创建一个定时器，一般的定时器的创建主要有三个步骤：首先创建定时器，然后编写 WM_TIMER 消息响应的代码，最后删除定时器。比如说，以下图为例，要模拟游戏人物的受到重击以后摔打的情景，就可以先创建一个定时器：

```
//建立定时器，间隔 0.09s 发出消息
    SetTimer(hwnd,1,90,NULL);
```

再在窗口过程函数中响应 WM_TIMER 消息：

```
switch (message)
{
    case WM_TIMER:
        Game_Paint(hwnd);
        break;
}
```

在 Game_Paint(HWND)函数中实现每一帧动作图的连续贴图，最后在程序结束的地方调用 KillTimer(hwnd,1);就可以模拟游戏人物的受到重击以后摔打的情景。

因为定时器是使用硬件定时器中断，有时会让人产生误解，认为程序会异步地中断来响应 WM_TIMER 消息。其实，WM_TIMER 消息并不是异步的。WM_TIMER 消息放在正常的消息队列中，和其他消息排列在一起，所以，在 SetTimer 中指定时间间隔为 1000 毫秒，不能保证程序每隔 1000 毫秒就会收到一个 WM_TIMER 消息。如果其他执行事件超过一秒，在此期间，程序将收不到任何 WM_TIMER 消息。所以利用这种定时器的技术来实现游戏动画显示效果是不准确的。一般而言，游戏本身需要显示顺畅的游戏画面，使玩家感觉不到延迟的状态。基本游戏画面必须在一秒钟之内至少更新 25 次以上，这一秒钟内程序还必须进行消息的处理和大量数学运算甚至音效的输出等操作。而使用定时器的消息来驱动这些操作，往往达不到所要求的标准，不然就会产生画面显示不顺畅和游戏响应时间太长的情况。游戏循环技术就不存在这个问题。

游戏循环是将原先程序中的消息循环加以修改，具体的实现方法就是判断其中的内容目前是否有要处理的消息，如果有则进行处理，否则按照设定的时间间隔来重绘画面。游戏循环技术的具体代码如下：

```
    //游戏循环
    while( msg.message!=WM_QUIT )
    {
        if( PeekMessage( &msg, NULL, 0,0 ,PM_REMOVE) )
          {
```

```
            TranslateMessage( &msg );
            DispatchMessage( &msg );
        }
        else
        {
            tNow = GetTickCount();
            if(tNow-tPre >= 100)
            Game_Paint(hdc);
        }
    }
```

当收到的 msg.message 不是窗口结束消息 WM_QUIT，则继续运行循环，其中 msg 是一个 MSG 的消息结构，其结构成员 message 则是一个消息类型的代号。使用 PeekMessage 函数来检测目前是否有需要处理的消息，若检测到消息（包含 WM_QUIT 消息）则会返回一个非“0”值，否则返回“0”。因此在游戏循环中，若检测到消息便进行消息的处理，否则运行 else 叙述之后的程序代码。需要注意的是，PeekMessage 函数不能用原先消息循环的条件 GetMessage 取代，因为 GetMessage 函数只有在取得 WM_QUIT 消息时才会返回"0",其他时候则是返回非“0”值或“-1”(发生错误时)。GetTickCount 函数会取得系统开始运行到目前所经过的时间，单位是毫秒 milliseconds。tPre 记录前次绘图的时间，而 tNow-tRre 则是计算上次绘图到这次循环运行之间相差多少时间。这里设置为若相差 40 个单位时间以上则再次进行绘图的操作，通过这个数值的控制可以调整游戏运行的速度。这里设定 40 个单位时间（微秒）的原因是，因为每隔 40 个单位进行一次绘图的操作，那么 1 秒钟大约重绘窗口 1000/40=25 次刚好可以达到期望值。由于循环的运行速度远比定时器发出时间信号来得要快，因此使用游戏循环可以更精准地控制程序运行速度并提高每秒钟画面重绘的次数。后面的游戏设计使用的也是游戏循环技术。

5. 游戏模型设计与实现

上一个章节中介绍了游戏实现所涉及到的核心技术，这里给出游戏模型的详细设计过程，主要包含人物行为、人物情感模型、物理建模、重力系统、碰撞检测、粒子系统、游戏 AI 以及游戏的起始和结束画面的设计等模块。这些模块所涉及到的资源均以位图的形式来存储，为了方便程序的实现，部分位图还进行了规格化的处理，设定每一帧的长度和宽度为固定值。

5.1 人物行为设计

在游戏开发中，人物的行为有一个术语，叫做游戏的**行为系统**。游戏的行为实系统际上是游戏内部运行机制决定的游戏的输入/输出集，它决定了游戏者在特定的游戏系统中可以做什么，不可以做什么。游戏行为系统的功能是作为情感释放手段，它也是游戏交互性的重要组成部分。

在此次设计的格斗游戏中，人物的行为自然是必不可少的东西。考虑到时间的有限性，此次的“街头霸王”格斗游戏对人物以及人物的行为进行了大量的简化，这里规定只有两个游戏人物，一个是 Hero，取材于真实《街头霸王》游戏中的人物 Ryu，另一个人物是 Robot，

也是取材于真实《街头霸王》游戏中的人物 Ryu，为了区分二者，在设计的时候我们给人物穿上不同的衣服。玩家可以控制 Hero，Robot 由电脑端控制。二者的行为设计也进行了大量的简化。规定 Hero 和 Robot 只有 6 种行为，分别为前进、后退、等待、拳击、踢腿、发绝招。前面三种行为属于游戏人物的常规行为，后面三种行为属于游戏正常进行中的格斗攻击行为。下面给出二者 6 种行为的具体设计以及实现。

5.1.1 Hero

Hero 的前进行为如下图：

因为 Hero 的前进是为了攻击，所以这里的每一帧身体都略微前倾，稍带有攻击的动作。注：为了方便后期的编程，这里统一设计每一帧的大小为 63*93，单位是像素。

Hero 的后退行为设计如下图：

考虑到 Hero 的后退行为是为了防止 Robot 的攻击，所以这里的每一帧都略微带有向后仰的动作，这样看起来有种防守的效果。这里每一帧的大小也是 63*93 像素。

Hero 等待行为设计如下图：

Hero 等待行为的设计是为了响应游戏初始化时，玩家没有进行任何操作的时候的动作状态，所以每一帧看起来几乎一样，但是为了增加动态效果，每一帧的人物动作也稍有不同，所以 Hero 即使处于等待状态时看起来也是左右摇摆的。这里每一帧的大小也是统一设置为 63*93 像素。

到目前为止，人物的常规行为已经设计完毕。具体的编程实现其实也比较简单。因为这三种行为都属于常规行为，所以可以用一个 HBITMAP 类型的数组来存储这些资源句柄。

```
HBITMAP g_hHeroDirection[3]={NULL};
```

为了方面识别 Hero 的行为，可以创建一个 int 类型的变量用来标识 Hero 的三种不同行为，0,1,2 分别表示 Hero 的三种行为。

```
int g_iHeroDirection;
```

声明完以后可以在游戏资源初始化中进行初始化。因为像这些位图资源、内存设备环境 DC、游戏人物的初始状态的设置等这些都是是必比可少的，所以统一写到一个 Game_Init() 函数中。可以在这个 Game_Init 函数中编写加载位图资源的代码和并将 g_iHeroDirection 设置为 2，表示处于等待的状态：

```
g_hHeroDirection[0]=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R1_GOBACK));
g_hHeroDirection[1]=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R1_GOFORWARD));
g_hHeroDirection[2]=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R1_WAIT));
g_iHeroDirection=2;//wait 状态
```

对于游戏人物的行为的绘制，我们可以统一写到一个 Game_Paint 函数中，

```
//Hero 贴图，针对其各个方向进行贴图
if(!g_gameOverFlag&&!g_heroActionFlag&&!g_heroBeAttactedLightFlag&&!g_heroBeAttactedHeavyFlag)
{
        SelectObject(g_bufdc,g_hHeroDirection[g_iHeroDirection]);
        TransparentBlt(g_hdcMem,
            g_iHeroX,g_iHeroY,
            bmpHero.bmWidth/6,bmpHero.bmHeight,
            g_bufdc,
            g_iHeroFileNum*bmpHero.bmWidth/6,0,
            bmpHero.bmWidth/6,bmpHero.bmHeight,
            RGB(255,255,255));
}
```

注：g_gameOverFlag 表示游戏是否结束，g_heroActionFlag 表示 Hero 是否处于攻击状态，g_heroBeAttactedLightFlag 表示 Hero 是否受到轻击的状态，g_heroBeAttactedHeavyFlag 表示 Hero 是否受到来自 Robot 的重击。只有三种状态全部没有发生情况下，才会绘制 Hero 的这三种常规状态。

下面给出 Hero 攻击行为的设计。Hero 的拳击行为设计如下图：

考虑到攻击状态较常规行为肢体长度发生变化，所以 Hero 攻击状态的行为位图统一设计为 116*93 像素。

Hero 的踢腿行为设计如下图，大小依然为 116*93 像素：

Hero 发绝招行为设计如下图，大小依然为 116*93 像素：

注：在真实的《街头霸王》游戏中 Ryu 有一个很经典的绝招就是推出一个波形的类似子弹的东西，这里也设计了一个蓝色的波形子弹，如下图，大小为 40*20 像素：

到目前为止，Hero 的行为设计就全部完成了。具体的编码实现，这部分稍微有一点复杂，因为考虑到前两个动作是常规的攻击行为，而最后发绝招的行为比较特殊，会伴随着一个波形子弹的产生，所以这个行为需要拿出来单独处理。

首先，定义一个表示游戏人物攻击行为的枚举，这里默认某一时刻游戏人物只能进行一种攻击行为，所以用枚举来定义。

```
//攻击动作枚举
enum ActionType
{
 ACTION_TYPE_BOXING=0,//拳击
 ACTION_TYPE_KICK=1,//踢腿
 ACTION_TYPE_WAVEBOXING=2,//绝招
 ACTION_TYPE_NULL=3,//无攻击动作
};
```

之所以加进去一个 ACTION_TYPE_NULL，因为游戏人物并不是时刻处于攻击状态的，所以用第四种状态来表示游戏人物不处于任何攻击状态。

然后再声明一个表示 Hero 攻击类型的变量，

```
//Hero 的出招类型
ActionType g_HeroActionType;
```

同时声明一个 bool 类型的变量用来判断 Hero 是否处于攻击状态。

```
bool g_heroActionFlag;
```

既然是攻击状态，双方都有可能受到攻击，因攻击的类型不同，这里规定拳击和踢腿属于轻的攻击，发绝招属于重的攻击。所以再声明两个 bool 变量表示 Hero 是否受到轻或者重程度的攻击：

```
bool g_heroBeAttactedLightFlag;
bool g_heroBeAttactedHeavyFlag;
```

在游戏资源初始化函数中除了加载行为位图，还要设置以上标志的值为 false：

```
g_hHeroHit[0]=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R1_BOXING));
g_hHeroHit[1]=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R1_KICK));
g_hHeroHit[2]=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R1_WAVEBOXING));
g_hHeroWave=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R1_WAVE));


g_heroActionFlag=false;
g_heroBeAttactedHeavyFlag=false;
g_heroBeAttactedLightFlag=false;
```

在 Game_Paint 函数中绘制人物攻击状态的行为，程序的结构也比较简单，主要思路如下：

```
if(游戏未结束&&Hero 未受到攻击)
{
 if(绝招攻击状态)
      {
       Hero 发绝招行为贴图;
       推出的波形子弹贴图;
       if(子弹打到对方)
       {
            if(对方体力值>0)
            {
                 造成伤害，对方体力值衰减;
```

```
                Robot 已被重击标志置为 true;
                }
                else//体力值小于等于 0
                {
                Robot 体力值置为 0;
                游戏结束标志置为 true;
                }
            }
        }
        else//普通招式
        {
         Hero 常规招式贴图;
         if(打到对方)
         {
              if(对方体力值>0)
              {
                   造成伤害，对方体力值衰减;
                   Robot 已被轻击标志置为 true;
                  }
                  else//体力值小于等于 0
                  {
                   Robot 体力值置为 0;
                   游戏结束标志置为 true;
                  }
        }
}
```

上面的游戏思路还是比较清晰的，根据上面的伪代码很容易写出下面的代码。

```
//Hero 攻击，招式控制，招式不为空，且处于攻击状态
if((g_HeroActionType!=ACTION_TYPE_NULL)&&g_heroActionFlag&&!g_gameOverFlag)
{
            ……
        if(g_HeroActionType==ACTION_TYPE_WAVEBOXING)
        {
            //人物招式贴图
            SelectObject(g_bufdc,g_hHeroHit[g_HeroActionType]);
            TransparentBlt(g_hdcMem,
                  g_iHeroX,g_iHeroY,
                  bmpHeroAction.bmWidth/6,bmpHeroAction.bmHeight,
                  g_bufdc,
                  g_iHeroFileNum*bmpHeroAction.bmWidth/6,0,
                  bmpHeroAction.bmWidth/6,bmpHeroAction.bmHeight,
                  RGB(255,255,255));
            //绝招效果贴图
            SelectObject(g_bufdc,g_hHeroWave);
```

```
        if (g_iHeroWavesNum!=0)
        {
            for (int i=0;i<10;i++)
                if (heroWaves[i].exist)
                {
                    TransparentBlt(g_hdcMem,
                        heroWaves[i].x+45,heroWaves[i].y,
                        bmpHeroWave.bmWidth,bmpHeroWave.bmHeight,
                        g_bufdc,
                        0,0,
                        bmpHeroWave.bmWidth,bmpHeroWave.bmHeight,
                        RGB(255,255,255));
                    heroWaves[i].x+=20;
                    //Hero 打出的波距离 Robot 小于 20 个像素就认为是打中
                    if((g_iRobotX-heroWaves[i].x)<20)
                    {
                        if(g_iRobotStrength>0)
                        {
                            g_robotBeAttactedHeavyFlag=true;
                            //造成 5-15 之间的伤害
                            float damage=5+(float)(rand()%10);
                            g_iRobotStrength-=damage;
                        }
                        else
                        {
                            g_iRobotStrength=0;
                            g_gameOverFlag=true;
                        }
                        g_RobotActionType=ACTION_TYPE_NULL;
                        g_iHeroWavesNum--;
                        heroWaves[i].exist=false;
                    }
                    //没打中到达窗口边缘自动消失
                    if (heroWaves[i].x>wndRect.right-bmpHeroWave.bmWidth)
                    {
                        g_iHeroWavesNum--;
                        heroWaves[i].exist=false;
                    }
                }
        }
    }
    else//普通招式贴图
    {
        if(!g_gameOverFlag)
```

```
            {
                SelectObject(g_bufdc,g_hHeroHit[g_HeroActionType]);
                TransparentBlt(g_hdcMem,
                    g_iHeroX,g_iHeroY,
                    bmpHeroAction.bmWidth/6,bmpHeroAction.bmHeight,
                    g_bufdc,
                    g_iHeroFileNum*bmpHeroAction.bmWidth/6,0,
                    bmpHeroAction.bmWidth/6,bmpHeroAction.bmHeight,
                    RGB(255,255,255));
                //位图差距为 116
                if((g_iRobotX-g_iHeroX)<=120)
                {
                    g_robotBeAttactedLightFlag=true;
                    if (g_iRobotStrength>0)
                    {
                        //造成 0-5 之间的伤害
                        float damage=(float)(rand()%5);
                        g_iRobotStrength-=damage;
                        Sleep(10);
                    }
                    else
                    {
                        g_iRobotStrength=0;
                        g_gameOverFlag=true;
                    }
                }
            }
        }
        //g_heroActionFlag=false;
    }
```

注：Hero 发出绝招的时候推出的波形子弹是放在一个结构体数组中的，结构体的设计如下：

```
//绝招时发出的子弹
struct WaveBullets
{
 int x,y;//波的坐标
 bool exist;
};
//heroWaves 存储 Hero 打出的波形子弹的信息
WaveBullets heroWaves[10];
//g_iHeroWavesNum 记录 Hero 的波形子弹的数目
int g_iHeroWavesNum=0;
```

其中，exist 用于标志子弹是否存在。当玩家按键盘的 L 键的时候就会触发这个动作，所以 VK_L 的消息响应代码中包含了波形子弹数组的初始化：

```
case VK_L:
g_heroActionFlag=true;
g_HeroActionType=ACTION_TYPE_WAVEBOXING;
//g_robotBeAttactedHeavyFlag=true;
for (int i=0;i<10;i++)
{
    if (!heroWaves[i].exist)
    {
        heroWaves[i].x=g_iHeroX+50;
        heroWaves[i].y=g_iHeroY+20;//大概处于出手位置，更显真实
        heroWaves[i].exist=true;
        g_iHeroWavesNum++;//波的数目累加
        break;
        }
    }
break;
```

其中，波形子弹的 x 和 y 坐标加了一定的数值是为了调整其刚开始的位置，因为在真实的游戏场景中，波形子弹是从手那里推出来的，所以微调使其接近手的位置。在上面的绝招效果贴图部分，通过实时遍历子弹数组，判断波形子弹的位置与 Robot 的位置，如果二者的距离之差小于 120 个像素的时候就可以认为打中了 Robot，当然波形子弹也要随之消失，所以这里也需要将波形子弹的 exist 属性置为 false，使其消失。至于伤害值如何处理，这个属于游戏 AI 的部分，后面会给出详细的介绍。

5.1.2 Robot

Robot 的取材也是真实《街头霸王》游戏中的 Ryu 这一角色，只是换了一身的灰色的衣服。他的行为与 Hero 的行为大致类似，分为 5 种。下面只给出设计图，至于如何实现的，这个属于游戏 AI 的范畴，后面会给出详细的介绍。

Robot 前进行为设计如下图，大小规格为 63*93 像素：

Robot 后退行为设计如下图，大小规格也是 63*93 像素：

Robot 等待行为设计如下图，大小规格也为 63*93 像素：

Robot 攻击行为在这里简化为 2 种，拳击、踢腿。主要是因为考虑到以下两点：一方面是因为这个部分加进去会使整个程序变得更加混乱。因为在设计 Hero 发大招的时候，波形子弹全部是通过标志位的方式来判断的，并没有用多线程技术。再添加 Robot 的绝招行为，需要设置更多的标志位，更容易引发不可预料的异常；另一方面是因为波形子弹是存在数组里面的，如果设计 Robot 也有发绝招的行为，就需要处理这种情况，某一时刻，两个人都在发绝招，这涉及到波形子弹的碰撞检测。数组的这种存储结构就决定了只能以遍历整个数组的方式来判断哪个波形子弹与另外一个波形子弹碰撞了，时刻计算二者之间的水平距离只差，这个比较耗时。而且 WinCE 的 CPU 处理速度本身就不快，很容易造成整个游戏界面来不及刷新。综合上述考虑，Robot 就不设计攻击的发绝招的攻击行为了。

Robot 的拳击行为设计如下图，规格为 116*93 像素：

Robot 的踢腿行为设计如下图，规格大小也是 116*93 像素：

5.2 人物情感模型设计

倘若一部游戏不能使游戏者获得某种深层的情感，那么它所受到的欢迎程度将是有限的。在确定了具有竞争性的游戏内部机制后，下一步需要考虑的就是游戏的情感世界。 在一个大型的游戏中，为了增加游戏的真实性，人物的情感设计也是不可或缺的一部分。在本游戏中也对人物的情感模型进行了设计。主要是以下几个方面：游戏人物受攻击以后的表情动作和游戏人物取得胜利以后的表情动作以及游戏人物阵亡以后的表情动作。

5.2.1 Hero

Hero 的主要表现是受到轻的攻击像拳击、踢腿以后的表情动作以及取得胜利或者阵亡以后的表情动作。Hero 受到轻攻击的表情设计如下，每一帧的大小像素是 63*93：

由于受到轻攻击的部位主要集中在头部，所以设计有一个头部后仰，面部抽搐的表情动

作。

Hero 取得胜利以后的表情动作设计如下，每一帧的规格统一为 63*93 像素：

Hero 阵亡以后的表情设计如下，每一帧的规格同样统一为 116*93 像素：

5.2.2 Robot

Robot 的主要表现是受到轻的攻击像拳击、踢腿以后的表情动作以及受到 Robot 发绝招以后的摔倒在地的表情动作以及阵亡的表情动作。Robot 受到轻攻击的表情设计如下，每一帧的大小像素是 63*93：

Robot 受到重攻击的表情设计如下，每一帧的大小像素是 116*93：

Robot 阵亡以后的表情设计如下，每一帧的像素为 116*93：

在具体的编程实现的时候，这些表情动作的实现其实也很简单，无非就是判断相应的标志位，加载各自的资源，以 Hero 受到轻击打为例：

```
//Hero 受到轻击打
if(!g_gameOverFlag&&g_heroBeAttactedLightFlag&&g_RobotActionType!=ACTION_TYPE_NULL)
{
    g_heroActionFlag=false;
    SelectObject(g_bufdc,g_hHeroBeAttactedLight);
    TransparentBlt(
        g_hdcMem,
        g_iHeroX,g_iHeroY,
        bmpHero.bmWidth/6,bmpHero.bmHeight,
```

```
            g_bufdc,
            g_iHeroFileNum*bmpHero.bmWidth/6,0,
            bmpHero.bmWidth/6,bmpHero.bmHeight,
            RGB(255,255,255));
        g_heroBeAttactedLightFlag=false;
}
```

其他的实现类似，这里就不贴代码了。

5.3 物理建模

在任意一款成功的游戏中，物理建模从来都是非常核心的部分，对于一些功能全面的商业游戏引擎都有着健壮而强大的代码负责引擎内部完善的物理建模。在本游戏的设计中，也涉及到了这部分，主要是在模拟人物和雪花的运动轨迹。这里主要采用匀速运动模型，物理中计算物体的位移有一个非常经典的公式，

$$s=s_0+v*t$$

将其进行正交方向分解成 X 和 Y 两个方向，于是就有下面两个公式：

$$s_x=s_{x0}+v_x*t$$

$$s_y=s_{y0}+v_y*t$$

基于这两个公式，就可以近似地模拟雪花的运动轨迹。在代码中换一种形式其实就是：

下次 X 轴坐标=当前 X 轴坐标+X 轴上的速度分量

下次 Y 轴坐标=当前 Y 轴坐标+Y 轴上的速度分量

具体编程实现时，人物的移动是通过响应键盘左右箭头消息来实现的：

```
//方向控制(WINCE 下不支持字母)
case VK_LEFT://后退
    if(!g_gameOverFlag)//游戏未结束
    {
        g_heroActionFlag=false;
        g_HeroActionType=ACTION_TYPE_NULL;
        g_robotBeAttactedLightFlag=false;
        g_robotBeAttactedHeavyFlag=false;
        g_iHeroX-=10;
        g_iHeroDirection=0;
        if (g_iHeroX<=0)
            g_iHeroX=0;
……….
    }
```

加粗的部分就是 Hero 的 X 方向上的位移运动模拟，雪花粒子的模拟也与此类似，在后面的粒子系统部分再做详细的介绍。其实**重力系统**的模拟也更加简单，无非就是在速度的基础上每次再加一个加速度的值，比如说后面雪花的下落，每次让其 Y 方向的速度增加一个 Y 方向的加速度值，重新计算一下 Y 方向上的位移就可以了，这里不再赘述。

5.4 碰撞检测设计

在成功的二维或者三维游戏中，碰撞检测(collision detection)的存在可以增强游戏场景的真实度，更加符合真实的自然规律。本游戏中，也有所涉及。主要是针对人物、人物发绝招的时候的推出的波形子弹以及雪花粒子的所处的范围的检测。比如说，游戏人物走到窗口边界以后不能继续移动；两个人物碰撞在一起的时候，应停止在当前位置；Hero 发绝招推出的波形子弹碰到 Robot 以后应当消失，以及雪花下落到底面以后也应当消失等等。

以 Hero 发绝招推出的波形子弹碰到 Robot 为例，检测实现思路很简单，就是实时判断波形子弹的当前位置的 X 坐标与 Robot 的当前位置 X 坐标只差，如果小于 20，就认为碰撞了，雪花的 exist 属性置为 false 即可，其他的标志位也进行相应的设置。具体的实现代码如下：

```
SelectObject(g_bufdc,g_hHeroWave);
if (g_iHeroWavesNum!=0)
{
    for (int i=0;i<10;i++)
        if (heroWaves[i].exist)
        {
            TransparentBlt(g_hdcMem,
                heroWaves[i].x+45,heroWaves[i].y,
                bmpHeroWave.bmWidth,bmpHeroWave.bmHeight,
                g_bufdc,
                0,0,
                bmpHeroWave.bmWidth,bmpHeroWave.bmHeight,
                RGB(255,255,255));
            heroWaves[i].x+=20;
            //Hero 打出的波距离 Robot 小于 20 个像素就认为是打中
            if((g_iRobotX-heroWaves[i].x)<20)
            {
                if(g_iRobotStrength>0)
                {
                    g_robotBeAttactedHeavyFlag=true;
                    //造成 5-15 之间的伤害
                    float damage=5+(float)(rand()%10);
                    g_iRobotStrength-=damage;
                }
                else
                {
                    g_iRobotStrength=0;
                    g_gameOverFlag=true;
                }
                g_RobotActionType=ACTION_TYPE_NULL;
                g_iHeroWavesNum--;
                heroWaves[i].exist=false;
            }
```

```
                    //没打中到达窗口边缘自动消失
                    if (heroWaves[i].x>wndRect.right-bmpHeroWave.bmWidth)
                    {
                        g_iHeroWavesNum--;
                        heroWaves[i].exist=false;
                    }
                }
            }
```

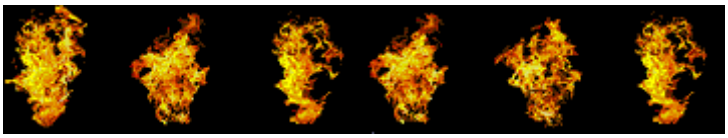
5.5 粒子系统设计

在游戏场景中，经常使用粒子系统模拟的现象有火、爆炸、烟、水流、火花、落叶、云、雾、雪、尘、流星尾迹或者象发光轨迹这样的抽象视觉效果，同样也可以增强游戏的真实感。在本游戏中，设计两种粒子，一种是雪花粒子，另外一种烈火燃烧的场景。雪花粒子的设计思路很简单，为了简化一下，这里雪花的实际存在就是一张设计好的 bmp 位图。程序运行时，会在窗口不断增加飞舞的雪花，当其数量达到一定的值的时候就不再增加。当雪花掉落到窗口底部的时候，就以随机的 X 坐标出现在窗口的顶部，这样就达到源源不断的雪花粒子下落的场景。具体编码实现也很简单：

```
//雪花粒子系统
struct Snow
{
    int x,y;//雪花的位置坐标
    BOOL exist;
};
//SnowFlowers 存储雪花粒子信息
Snow SnowFlowers[100];
//g_SnowNum 记录雪花的数目
int g_SnowNum=0;
//创建雪花粒子系统
    if (g_SnowNum<100)
    {
        SnowFlowers[g_SnowNum].x=rand()%wndRect.right;
        SnowFlowers[g_SnowNum].y=0;
        SnowFlowers[g_SnowNum].exist=true;
        g_SnowNum++;
    }
    //判断粒子是否存在，存在则进行透明贴图操作
    for (int i=0;i<100;i++)
    {
        if (SnowFlowers[i].exist)
        {
            //贴上粒子图
            SelectObject(g_bufdc,g_hSnow);
            TransparentBlt(g_hdcMem,
```

```
                SnowFlowers[i].x,SnowFlowers[i].y,
                bmpSnow.bmWidth,bmpSnow.bmHeight,
                g_bufdc,
                0,0,
                bmpSnow.bmWidth,bmpSnow.bmHeight,
                RGB(0,0,0));
            //0.5 的概率随机决定横向的移动方向和偏移量
            if (rand()%2==0)
                SnowFlowers[i].x+=rand()%6;
            else
                SnowFlowers[i].x-=rand()%6;
            //纵方向上做匀速运动
            SnowFlowers[i].y+=10;
            if (SnowFlowers[i].y>wndRect.bottom)
            {
                SnowFlowers[i].x=rand()%wndRect.right;
                SnowFlowers[i].y=0;
            }
        }
    }
```

本游戏的另一个粒子的设计就是烈火的燃烧，设计思路就是在路两边生火，毕竟游戏背景是黑夜，打架也不能瞎打。烈火粒子的实现就是连续加载设计好的 bmp 位图，设计如下，每一帧的大小规格是 60*64 像素：



由于这个场景一直存在，所以直接写在 Game_Paint 中就行了，两行代码：

```
//烈火燃烧画面
SelectObject(g_bufdc,g_hFire);
TransparentBlt(g_hdcMem,
    0,wndRect.right/3,
    bmpFire.bmWidth/6,bmpFire.bmHeight,
    g_bufdc,
    g_iHeroFileNum*bmpFire.bmWidth/6,0,
    bmpFire.bmWidth/6,bmpFire.bmHeight,
    RGB(0,0,0));
TransparentBlt(g_hdcMem,
    wndRect.right/20*19,wndRect.right/3,
    bmpFire.bmWidth/6,bmpFire.bmHeight,
    g_bufdc,
    (g_iHeroFileNum)*bmpFire.bmWidth/6,0,
    bmpFire.bmWidth/6,bmpFire.bmHeight,
    RGB(0,0,0));
```

5.6 游戏 AI 设计

游戏的 AI 也是一款成功的游戏不可缺少的一部分。AI 是人工智能(Artificial Intelligence)的缩写。这里所谓的游戏人工智能也只是渊博的人工智能领域的冰山一角，本游戏的 AI 设计并没有涉及类似于神经网络、基因算法、模糊数学等复杂的人工智能理论。由于本游戏实现的是人机对打，所以电脑端的人物自然就少不了 AI。这里实现也很简单，粗暴地设计当 Hero 向 Robot 进攻的时候，Robot 就立即后退闪躲；Hero 后退的时候，Robot 就主动快速发起进攻，这里位移是在同一步调的基础上加了一个随机值，用随机数进行模拟。对于 Robot 每次打出的伤害值的设计也是用随机数模拟。具体的编程实现如下：

```
//方向控制(测试发现 WINCE 下不支持字母)
    case VK_LEFT://后退
        if(!g_gameOverFlag)//游戏未结束
        {
            g_heroActionFlag=false;
            g_HeroActionType=ACTION_TYPE_NULL;
            g_robotBeAttactedLightFlag=false;
            g_robotBeAttactedHeavyFlag=false;
            g_iHeroX-=10;
            g_iHeroDirection=0;
            if (g_iHeroX<=0)
                g_iHeroX=0;

            g_iRobotDirection=1;//Hero 后退则 Robot 前进
            if (g_iRobotX>(g_iHeroX+g_bmpHeroHit.bmWidth/6))//未靠近 Hero
            {
                g_iRobotX-=(10+rand()%5);//以 10-15 的像素向 Hero 靠近
            }
            else
            {
                g_iRobotX=(g_iHeroX+g_bmpHeroHit.bmWidth/6);
            }
        }
        else//游戏结束
        {
            g_iHeroDirection=2;
            g_heroActionFlag=false;
            g_HeroActionType=ACTION_TYPE_NULL;
        }
        break;

    case VK_RIGHT://前进
        if (!g_gameOverFlag)//游戏未结束
        {
            g_heroActionFlag=false;
```

```
                g_robotBeAttactedLightFlag=false;
                g_robotBeAttactedHeavyFlag=false;
                g_iHeroX+=10;
                g_iHeroDirection=1;
                if(g_iHeroX>=(g_iRobotX-g_bmpRobotHit.bmWidth/6))//Hero 靠近 Robot 则不能再靠近
                    g_iHeroX=g_iRobotX-g_bmpRobotHit.bmWidth/6;
                //g_RobotActionType=ACTION_TYPE_KICK;

                g_iRobotDirection=0;//Hero 前进则 Robot 回退
                g_iRobotX+=(10+rand()%5);
                if(g_iRobotX>=(rect.right-116))
                    g_iRobotX=rect.right-116;
            }
            else//游戏结束
            {
                g_iHeroDirection=2;
                g_heroActionFlag=false;
                g_HeroActionType=ACTION_TYPE_NULL;
            }
            break;


//Robot 攻击
    if(!g_gameOverFlag&&g_robotActionFlag/*&&(g_heroActionFlag==false)*/)
    {
        SelectObject(g_bufdc,g_hRobotHit[g_RobotActionType]);
        TransparentBlt(g_hdcMem,
            g_iRobotX,g_iRobotY,
            g_bmpRobotHit.bmWidth/6,g_bmpRobotHit.bmHeight,
            g_bufdc,
            g_iRobotFileNum*g_bmpRobotHit.bmWidth/6,0,
            g_bmpRobotHit.bmWidth/6,g_bmpRobotHit.bmHeight,
            RGB(255,255,255));

        if((g_iRobotX-g_iHeroX)<=110)
        {
            g_heroBeAttactedLightFlag=true;
            if (g_iHeroStrength>0)
            {
                //造成 5-10 之间的伤害
                float damage=(float)(5+rand()%5);
                g_iHeroStrength-=damage;
                //Sleep(10);
            }
```

```
            else
            {
                g_iHeroStrength=0;
                g_gameOverFlag=true;
            }
        }
    }
```

5.7 其他

到目前为止整个游戏的框架已经搭出来了，当然为了更美观一些，在这里增加了开机画面和结束画面，以及游戏人物血量图的实时绘制。

5.7.1 游戏开机画面

开机画面设计得比较简陋，就是如下的一张图：

加了键盘控制，P 键开始游戏，Q 键退出游戏。实现也很简单，设置一个全局变量 first_start，初始值置为 true，一旦按了 P 键，其值置为 false，同时在调用 Game_Paint 函数的地方加一个 first_start 的值的检查即可。这里为了实现开机画面类似卷轴的效果，用了一个 for 循环，按行来加载开机画面。具体编码实现如下：

```
if (first_start)
{
    SelectObject(start_hdc,g_hStart);
    for (int i=0;i<bmpLoading.bmWidth;i++)
    {
        BitBlt(g_hdc,
            (wndRect.right-bmpLoading.bmWidth)/2,(wndRect.bottom-bmpLoading.bmHeight)/2+i,
            bmpLoading.bmWidth,1,
```

```
                start_hdc,0,i,SRCCOPY);
            Sleep(10);
        }
    }
```

5.7.2 游戏结束画面

游戏结束画面设计也很简单，就是显示“Game Over”的字样，如下图：

注：这是一张遮罩图，两张图拼在一起正好实现遮罩的效果。实现透明遮罩的思路也很简单，两次调用 BitBlt 函数。一次进行与操作，一次进行或操作，就可以达到背景透明的效果。具体编程实现如下：

```
//游戏结束画面加载
SelectObject(g_bufdc,g_hTryAgain);
BitBlt(g_hdcMem,
    wndRect.right/4,0,
    bmpTryAgain.bmWidth/2,bmpTryAgain.bmHeight,
    g_bufdc,
    bmpTryAgain.bmWidth/2,0,SRCAND);
BitBlt(g_hdcMem,
    wndRect.right/4,0,
    bmpTryAgain.bmWidth/2,bmpTryAgain.bmHeight,
    g_bufdc,
    0,0,SRCPAINT);
```

5.7.3 游戏人物血量图的绘制

格斗游戏中，每个人都有一个相应体力值，即对应着血量。实现动态加载每个人物的血量图其实也很简单。游戏设定人物的最大体力值为 100，格斗过程中，每个人人物的体力值会随着对方打出的伤害值衰减，按照每个人的体力值占最大体力值的比例动态计算需要加载的血量位图的宽度即可。其中，血量图由三部分组成，中间 KO logo 和各自的鲜红血量图，设计如下图：

为了将上面三个图片放在合适的位置上，起始的贴图位置需要进行适当地调整。比如说 Hero 的血量图在左侧，加载器血量图的时候，贴图宽度应为窗口宽度的一半再减去 KO logo 资源句柄宽度的一半。同时 KO logo 贴图的起始位置为窗口宽度的一半再减去其本身资源句柄宽度的一半。Robot 的血量图在窗口的右边，血量图的贴图宽度应为窗口宽度的一半再减去 KO logo 资源句柄宽度的一半。具体编程实现如下：

```
//血量位图中间的 KO logo 贴图
SelectObject(g_bufdc,g_hLogoKO);
StretchBlt(g_hdcMem,
    wndRect.right/2-g_bmpKO.bmWidth/2,0,
    g_bmpKO.bmWidth,g_bmpKO.bmHeight,
    g_bufdc,
    0,0,
    g_bmpKO.bmWidth,g_bmpKO.bmHeight,
    SRCCOPY);
//贴上 Hero 血量图，根据体力值实时计算贴图位置
SelectObject(g_bufdc,g_hHeroBlood);
StretchBlt(g_hdcMem,
(int)(wndRect.right/2*(float)(1.0-(float)(g_iHeroStrength/CHARACTER_MAX_STRENGTH))),0,
(int)((float)(g_iHeroStrength/CHARACTER_MAX_STRENGTH)*wndRect.right/2)-g_bmpKO.bmWidth/2,
g_bmpHeroBlood.bmHeight,
    g_bufdc,
    0,0,
(int)(g_bmpHeroBlood.bmWidth*(float)(g_iHeroStrength/CHARACTER_MAX_STRENGTH)),g_bmpHeroBlood.bmHeight,
    SRCCOPY);
//贴上 Robot 血量图，根据体力值实时计算贴图起始位置
SelectObject(g_bufdc,g_hRobotBlood);
StretchBlt(g_hdcMem,
    wndRect.right/2+g_bmpKO.bmWidth/2,0,
(int)(wndRect.right/2*(float)(g_iRobotStrength/CHARACTER_MAX_STRENGTH)),g_bmpRobotBlood.bmHeight,
    g_bufdc,
    0,0,
    g_bmpRobotBlood.bmWidth,g_bmpRobotBlood.bmHeight,
    SRCCOPY);
```

6. 游戏测试

游戏开机画面卷轴效果：

Hero 拳击：

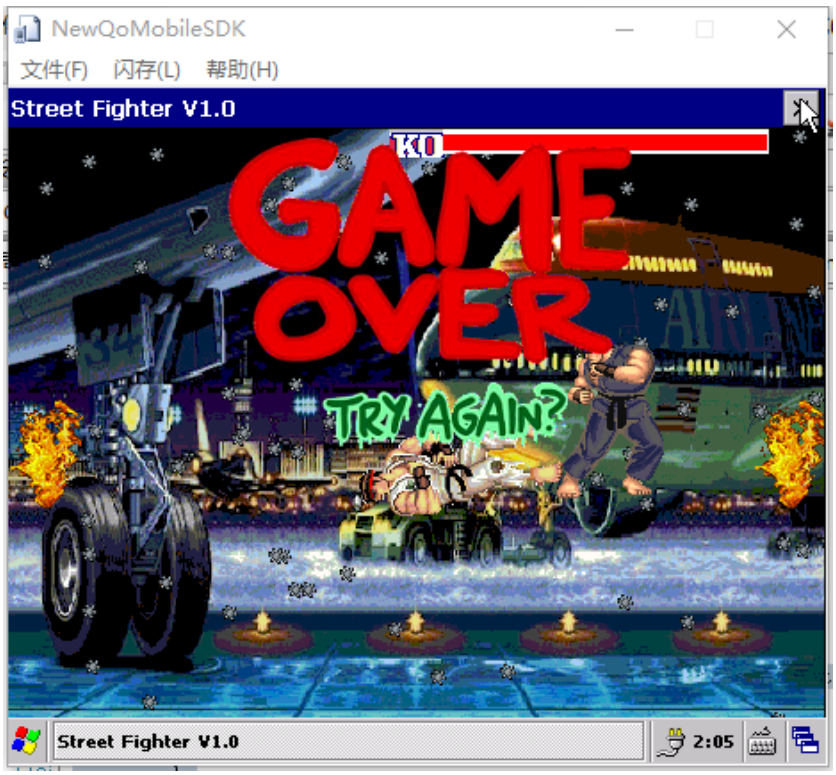
Hero 踢腿：

Hero 发绝招：

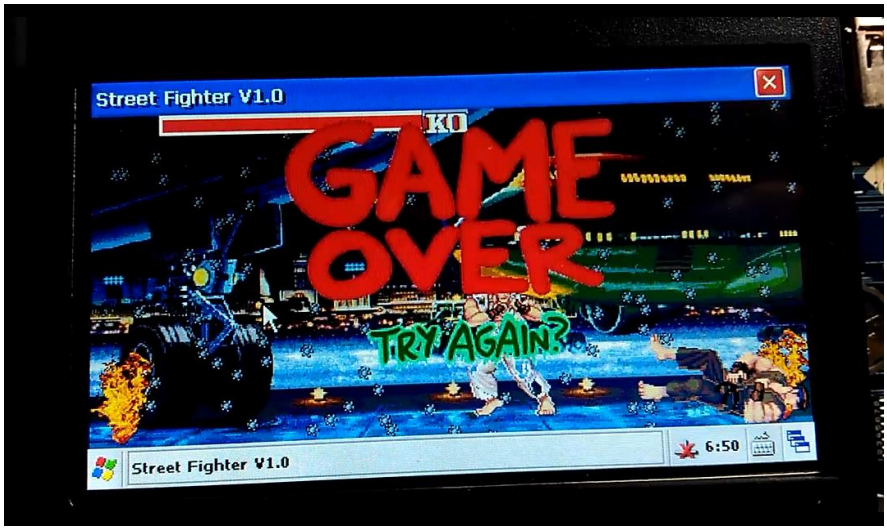
Robot 攻击：

Robot 阵亡：

Hero 阵亡：



游戏结束画面：



7. 个人总结

定期总结是一个好习惯。这里附上各个成员的心得体会。

7.1 李坚松

从确定游戏这个选题开始，我就有意识的开始接触关于游戏开发的东西。本来是想做三

维的游戏开发，测试后发现 WinCE 还不支持 DirectX 技术，于是转向学习 GDI 函数的使用和一些最基本的 Windows API 的使用，开始构思二维的实现。前期的学习再加上熟悉这些 API 的使用，大概用了一周的时间。之后又花了一天的时间寻找游戏图片素材，再自己进行设计并手工制作人物模型，这个过程比较漫长耗时，因为 Forlinx 的显示屏的尺寸较小，为了达到最佳的显示效果，曾进行了多次的试验。最终将游戏模型每一帧的规格统一为 63*93 和 116*93 两种。然后就开始着手整个游戏的编码实现，不幸的是，游戏的“闪屏”问题困扰了自己相当一段时间，一度怀疑是 CPU 的刷新频率太慢，也曾想过放弃开发游戏，不过最终还是找到一个比较有效的技术方案——“三缓冲”技术。此外，真机中的演示效果总是比模拟器中来得更加真实一点，尤其是人物的高度位置，在真机中游戏人物正好站在路面上。这次实验，从感性的方面来说，自己解决问题的能力确实提高了不少。从理性的角度来看，整个游戏从设计再到最终编码的实现，一整个流程走下来，让我对游戏的开发多多少少也有了一些高屋建瓴的认识。

当然，也有很多不足之处。游戏的代码写得比较烂，全部是过程式的，没用面向对象的方式编写，所以看起来难免有点凌乱。而且，人物绝招的波形子弹没有用多线程技术，所以两个人物同时发绝招的波形子弹的碰撞检测问题实现起来比较困难，容易造成游戏画面的卡顿。此次的实验作品仅仅是一个 Demo 版本，如果继续沿着这个思路走下去，再添加几个模块，还是可以开发出一个完整版的《街头霸王》的。

7.2 李标

此次，我们的实验是在飞凌嵌入式平台上，这与我们在实验室里用 C 开发不同，我们首先要熟悉 WinCE 开发，做一些准备工作，这给我们实验的前期的开发带来了一些困难。但是也是通过这次在 WinCE 上的开发挑战，我们学到了关于图形图片的处理，这给我们枯燥的开发带来了一些乐趣。实验虽然最终完成了，但是 WinCE 的开发我们要学的东西还有很多，此次实验锻炼了我们的能力同时，也让我们对嵌入式的开发更加感兴趣了。

7.3 童立成

本次试验前期立题之后部分时间用于素材寻找与剪切，确定合适位图大小与游戏场景布置，确定人物攻击，等待受击等模式。两个人物我们选择侧重玩家主体 Hero，并为其添加了绝招的设计，由于技术实现没有设计跳跃等躲避动作，但丰富了人物情感模型，对于轻击和重击会作出不同的反应。具体实现过程中出现了频率不对等问题，多亏坚松同学一步步的调试，目前整理到了一个合适的状态。此次试验涉及的部分较多，也给我们提供了很好的设计游戏的思路，为以后类似工作提供了良好的铺垫。

8. 参考文献

[1] Douglas Boling. Windows CE 6.0 开发者参考[M].北京:机械工业出版社,2009,3.

[2] 郑阿奇.DirectX 3D 游戏编程实用教程[M].北京:电子工业出版社,2011,2.

[3] 丁展.Visual C++游戏开发技术与实例[M].北京:人民邮电出版社,2005,2.

[4] 飞凌嵌入式官网. http://www.forlinx.com/ ,2015,12.

[5] 独立游戏大电影. http://www.iqiyi.com/w_19rtfq52pl.html ,2015,12.

9. 附录

下面贴上核心的代码，包括主要的结构体设计和核心函数的实现。

9.1 主要结构体

```
//动作枚举
enum ActionType
{
 ACTION_TYPE_BOXING=0,//拳击
 ACTION_TYPE_KICK=1,//踢腿
 ACTION_TYPE_WAVEBOXING=2,//绝招
 ACTION_TYPE_NULL=3,//无攻击动作
};
 //绝招时发出的子弹
struct WaveBullets
{
 int x,y;//波的坐标
 bool exist;
};
 //雪花粒子系统
struct Snow
{
 int x,y;//雪花的位置坐标
 BOOL exist;
};
```

9.2 核心函数

WinMain 主程序的入口点：

```
int WINAPI WinMain( __in HINSTANCE hInstance,
                    __in_opt HINSTANCE hPrevInstance,
                    __in_opt LPWSTR lpCmdLine,
                    __in int nShowCmd )
{
    //1.创建一个窗体类
    WNDCLASS ws;
    g_hInst=hInstance;
```

```
    ws.cbClsExtra    = 0;
    ws.cbWndExtra    = 0;
    ws.hbrBackground = (HBRUSH)GetStockObject(GRAY_BRUSH);
    ws.hCursor       = NULL;
    ws.hIcon         = NULL;
    ws.hInstance     = hInstance;
    ws.lpfnWndProc = WndProc;
    ws.lpszClassName = TEXT("Hello");
    ws.lpszMenuName   = NULL;
    ws.style         = CS_VREDRAW | CS_HREDRAW;

    //2.注册窗体类
    if (! RegisterClass(&ws))
        return -1;
    //3.创建窗体

    UINT sysWidth=GetSystemMetrics(SM_CXSCREEN);
    UINT sysHeight=GetSystemMetrics(SM_CYSCREEN);
    HWND hwnd = CreateWindow(TEXT("Hello"),TEXT("Street Fighter V1.0"),WS_VISIBLE |
WS_BORDER | WS_SYSMENU /*| WS_MINIMIZEBOX*/ /*| WS_MAXIMIZEBOX*/ | WS_CAPTION,
        0,0,sysWidth,sysHeight,
        NULL,NULL,hInstance,NULL);

    //4.更新窗体内容
    UpdateWindow(hwnd);
    ShowWindow(hwnd,nShowCmd);

    if (!Game_Init (hwnd))
    {
        MessageBox(hwnd, L"资源初始化失败", L"消息窗口", 0);
        return FALSE;
    }

    MSG msg={0};
    //5.获取系统消息
    while(msg.message!=WM_QUIT)
    {
        if (PeekMessage(&msg,0,0,0,PM_REMOVE))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
        else
        {
```

```
                g_tHeroNow=GetTickCount();//获取当前系统时间
                if ((g_tHeroNow-g_tHeroPre>=50)/*&&!first_start*/)
                {
                    Game_Paint(hwnd);
                }
            }
        }
        return 1;
}
```

窗口过程函数：

```
LRESULT CALLBACK WndProc(HWND hwnd,UINT message,WPARAM wParam,LPARAM lParam)
{
        RECT rect;
        GetClientRect(hwnd,&rect);


        switch (message)
        {
            //      case WM_TIMER:
            //          Game_Paint(hwnd);
            //          break;
        case WM_KEYDOWN:
            switch (wParam)
            {
            case VK_P:
                //开始游戏
                //first_start=false;
                break;
            case VK_Q:
                //退出游戏
                DestroyWindow(hwnd);
                PostQuitMessage(0);
                break;
            case VK_ESCAPE:
                DestroyWindow(hwnd);
                PostQuitMessage(0);
                break;

            //方向控制(测试发现 WINCE 下不支持字母)
            case VK_LEFT://后退
                if(!g_gameOverFlag)//游戏未结束
                {
                    g_heroActionFlag=false;
```

```
            g_HeroActionType=ACTION_TYPE_NULL;
            g_robotBeAttactedLightFlag=false;
            g_robotBeAttactedHeavyFlag=false;
            g_iHeroX-=10;
            g_iHeroDirection=0;
            if (g_iHeroX<=0)
                  g_iHeroX=0;

            g_iRobotDirection=1;//Hero 后退则 Robot 前进
            if (g_iRobotX>(g_iHeroX+g_bmpHeroHit.bmWidth/6))//未靠近 Hero
            {
                  g_iRobotX-=(10+rand()%5);//以 10-15 的像素向 Hero 靠近
            }
            else
            {
                  g_iRobotX=(g_iHeroX+g_bmpHeroHit.bmWidth/6);
            }
      }
      else//游戏结束
      {
            g_iHeroDirection=2;
            g_heroActionFlag=false;
            g_HeroActionType=ACTION_TYPE_NULL;
      }
      break;

case VK_RIGHT://前进
      if (!g_gameOverFlag)//游戏未结束
      {
            g_heroActionFlag=false;
            g_robotBeAttactedLightFlag=false;
            g_robotBeAttactedHeavyFlag=false;
            g_iHeroX+=10;
            g_iHeroDirection=1;
            if(g_iHeroX>=(g_iRobotX-g_bmpRobotHit.bmWidth/6))//Hero 靠近 Robot 则不能再靠近
                  g_iHeroX=g_iRobotX-g_bmpRobotHit.bmWidth/6;
            //g_RobotActionType=ACTION_TYPE_KICK;

            g_iRobotDirection=0;//Hero 前进则 Robot 回退
            g_iRobotX+=(10+rand()%5);
            if(g_iRobotX>=(rect.right-116))
                  g_iRobotX=rect.right-116;
      }
      else//游戏结束
```

```
            {
                g_iHeroDirection=2;
                g_heroActionFlag=false;
                g_HeroActionType=ACTION_TYPE_NULL;
            }
            break;

        //招式控制
        case VK_J:
            g_heroActionFlag=true;
            g_HeroActionType=ACTION_TYPE_BOXING;
            //g_RobotActionType=ACTION_TYPE_KICK;
            break;
        case VK_K:
            g_heroActionFlag=true;
            g_HeroActionType=ACTION_TYPE_KICK;
            //g_RobotActionType=ACTION_TYPE_BOXING;
            break;
        case VK_L:
            g_heroActionFlag=true;
            g_HeroActionType=ACTION_TYPE_WAVEBOXING;
            //g_robotBeAttactedHeavyFlag=true;
            for (int i=0;i<10;i++)
            {
                if (!heroWaves[i].exist)
                {
                    heroWaves[i].x=g_iHeroX+50;
                    heroWaves[i].y=g_iHeroY+20;//大概处于出手位置，更显真实
                    heroWaves[i].exist=true;
                    g_iHeroWavesNum++;//波的数目累加
                    break;
                }
            }
            break;
        }
        break;
case WM_DESTROY:
        Game_CleanUp(hwnd);
        PostQuitMessage( 0 );
        break;
default:
        //g_heroActionFlag=false;
        //g_robotActionFlag=true;
        g_iRobotDirection=2;//wait 防守状态
```

```
            g_iHeroDirection=2;//wait 的动作，处于防守状态
            return DefWindowProc(hwnd,message,wParam,lParam);
        }
        return 0;
}

/*
人物的绘制采用三缓冲技术,从 g_bufdc 贴图到 g_hdcMem，再进行透明化处理，
再将最后的结果显示在窗口 g_hdc 中
经比对，使用三缓冲可以有效地避免动画的闪烁
*/
//游戏资源初始化函数
BOOL Game_Init( HWND hwnd )
{
        HBITMAP wndBmp;

        g_hdc=GetDC(hwnd);
        //先创建内存 DC，再创建一个缓冲 DC
        g_hdcMem=CreateCompatibleDC(g_hdc);
        g_bufdc=CreateCompatibleDC(g_hdc);

        RECT wndRect;//窗体矩形
        GetClientRect(hwnd,&wndRect);

        //建一个和窗口大小相同的空的位图对象
        wndBmp=CreateCompatibleBitmap(g_hdc,wndRect.right,wndRect.bottom);
        //将空的位图放到 g_hdcMem 中
        SelectObject(g_hdcMem,wndBmp);

        //加载各种位图资源
        g_hBackGround=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_BKGRD));
        //g_hBlood=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_BLOOD));
        g_hHeroDirection[0]=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R1_GOBACK));
        g_hHeroDirection[1]=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R1_GOFORWARD));
        g_hHeroDirection[2]=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R1_WAIT));
        g_hHeroHit[0]=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R1_BOXING));
        g_hHeroHit[1]=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R1_KICK));
        g_hHeroHit[2]=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R1_WAVEBOXING));
        g_hHeroWave=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R1_WAVE));
        g_hHeroBeAttactedLight=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R1_BEATTACTED_LIGHT)
);
        g_hHeroBeAttactedHeavy=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R1_BEATTACTED_HEAV
Y));
        g_hHeroBlood=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R1_BLOOD));
```

```
        g_hHeroDie=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R1_DIE));
        g_hHeroWin=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R1_WIN));

        g_hRobotDirection[0]=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R2_GOBACK));
        g_hRobotDirection[1]=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R2_GOFORWARD));
        g_hRobotDirection[2]=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R2_WAIT));

        g_hRobotHit[0]=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R2_KICK));
        g_hRobotHit[1]=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R2_BOXING));
        g_hRobotHit[2]=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R2_WAVEBOXING));
        g_hRobotWave=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R2_WAVE));
        g_hRobotBeAttactedHeavy=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R2_BEATTACTED_HEAV
Y));
        g_hRobotBeAttactedLight=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R2_BEATTACTED_LIGHT
));
        g_hRobotBlood=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R2_BLOOD));
        g_hRobotDie=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_R2_DIE));

        g_hLogoKO=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_KO));
        g_hLoading=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_LOADING));
        g_hStart=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_START));
        g_hSnow=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_SNOW));
        g_hFire=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_FIRE));
        g_hTryAgain=LoadBitmap(g_hInst,MAKEINTRESOURCE(IDB_TRY_AGAIN));

        GetObject(g_hHeroHit[0],sizeof(g_bmpHeroHit),&g_bmpHeroHit);
        GetObject(g_hRobotHit[0],sizeof(g_bmpRobotHit),&g_bmpRobotHit);
        GetObject(g_hHeroBlood,sizeof(g_bmpHeroBlood),&g_bmpHeroBlood);
        GetObject(g_hRobotBlood,sizeof(g_bmpRobotBlood),&g_bmpRobotBlood);
        GetObject(g_hLogoKO,sizeof(g_bmpKO),&g_bmpKO);

        //Hero 贴图的起始坐标
        g_iHeroX=wndRect.right/4;
        g_iHeroY=wndRect.right/4;
        g_iHeroDirection=2;//wait 状态
        g_HeroActionType=ACTION_TYPE_NULL;
        g_heroActionFlag=false;
        g_heroBeAttactedHeavyFlag=false;
        g_heroBeAttactedLightFlag=false;
        g_iHeroStrength=100.0;//Hero 体力值初始化为 100
        g_iHeroFileNum=0;

        g_iRobotFileNum=0;
        //Robot 贴图的起始坐标
```

```
	g_iRobotX=wndRect.right/4*3;
	g_iRobotY=wndRect.right/4;
	g_iRobotDirection=2;
	g_RobotActionType=ACTION_TYPE_NULL;
	g_robotActionFlag=false;
	g_robotBeAttactedHeavyFlag=false;
	g_robotBeAttactedLightFlag=false;
	g_iRobotStrength=100.0;//Robot 体力值初始化为 100

	g_gameOverFlag=false;

	BITMAP bmpLoading;
	GetObject(g_hStart,sizeof(bmpLoading),&bmpLoading);
	//加载开始画面
	first_start=true;
	HDC start_hdc=NULL;
	start_hdc=CreateCompatibleDC(g_hdc);
	if (first_start)
	{
		SelectObject(start_hdc,g_hStart);
		for (int i=0;i<bmpLoading.bmWidth;i++)
		{
			BitBlt(g_hdc,
				(wndRect.right-bmpLoading.bmWidth)/2,(wndRect.bottom-bmpLoading.bmHeight)/2+i,
				bmpLoading.bmWidth,1,
				start_hdc,0,i,SRCCOPY);
			Sleep(10);
		}
	}
	g_iFireFrameNum=0;
	Game_Paint(hwnd);
	return TRUE;
}
//游戏画面的绘制函数
VOID Game_Paint(HWND hwnd)
{
	RECT wndRect;
	BITMAP bmpBkGrd;
	//获取背景图片的大小
	GetObject(g_hBackGround,sizeof(bmpBkGrd),&bmpBkGrd);
	//获取窗口尺寸
	GetClientRect(hwnd,&wndRect);

	BITMAP bmpHero;//hero 位图信息 63*93
```

```
GetObject(g_hHeroDirection[0],sizeof(bmpHero),&bmpHero);
BITMAP bmpRobot;//robot 位图信息,63*93
GetObject(g_hRobotDirection[0],sizeof(bmpRobot),&bmpRobot);
//开始界面
BITMAP bmpStart;
GetObject(g_hStart,sizeof(bmpStart),&bmpStart);
BITMAP bmpSnow;
//获取雪花位图的大小
GetObject(g_hSnow,sizeof(bmpSnow),&bmpSnow);
BITMAP bmpFire;
GetObject(g_hFire,sizeof(bmpFire),&bmpFire);
BITMAP bmpHeroWin;
GetObject(g_hHeroWin,sizeof(bmpHeroWin),&bmpHeroWin);
BITMAP bmpTryAgain;
GetObject(g_hTryAgain,sizeof(bmpTryAgain),&bmpTryAgain);

//绘制开始界面
//MessageBox(hwnd, L"是否开始游戏？", L"消息窗口", 0);

//先贴上背景图,将其存入内存 DC g_hdcMem 中
SelectObject(g_bufdc,g_hBackGround);
//拉伸的方式全屏贴背景图
StretchBlt(g_hdcMem,
        0,0,
        wndRect.right,wndRect.bottom,
        g_bufdc,0,0,
        bmpBkGrd.bmWidth,bmpBkGrd.bmHeight,
        SRCCOPY);
//火焰燃烧画面
SelectObject(g_bufdc,g_hFire);
TransparentBlt(g_hdcMem,
        0,wndRect.right/3,
        bmpFire.bmWidth/6,bmpFire.bmHeight,
        g_bufdc,
        g_iHeroFileNum*bmpFire.bmWidth/6,0,
        bmpFire.bmWidth/6,bmpFire.bmHeight,
        RGB(0,0,0));
TransparentBlt(g_hdcMem,
        wndRect.right/20*19,wndRect.right/3,
        bmpFire.bmWidth/6,bmpFire.bmHeight,
        g_bufdc,
        (g_iHeroFileNum)*bmpFire.bmWidth/6,0,
        bmpFire.bmWidth/6,bmpFire.bmHeight,
        RGB(0,0,0));
```

```
//创建雪花粒子系统
if (g_SnowNum<100)
{
	SnowFlowers[g_SnowNum].x=rand()%wndRect.right;
	SnowFlowers[g_SnowNum].y=0;
	SnowFlowers[g_SnowNum].exist=true;
	g_SnowNum++;
}
//判断粒子是否存在，存在则进行透明贴图操作
for (int i=0;i<100;i++)
{
	if (SnowFlowers[i].exist)
	{
		//贴上粒子图
		SelectObject(g_bufdc,g_hSnow);
		TransparentBlt(g_hdcMem,
			SnowFlowers[i].x,SnowFlowers[i].y,
			bmpSnow.bmWidth,bmpSnow.bmHeight,
			g_bufdc,
			0,0,
			bmpSnow.bmWidth,bmpSnow.bmHeight,
			RGB(0,0,0));
		//随机决定横向的移动方向和偏移量
		if (rand()%2==0)
			SnowFlowers[i].x+=rand()%6;
		else
			SnowFlowers[i].x-=rand()%6;
		//纵方向上做匀速运动
		SnowFlowers[i].y+=10;
		if (SnowFlowers[i].y>wndRect.bottom)
		{
			SnowFlowers[i].x=rand()%wndRect.right;
			SnowFlowers[i].y=0;
		}
	}
}

BITMAP bmpHeroWave;
GetObject(g_hHeroWave,sizeof(bmpHeroWave),&bmpHeroWave);
//Hero 攻击，招式控制，招式不为空，且处于攻击状态
if((g_HeroActionType!=ACTION_TYPE_NULL)&&g_heroActionFlag&&!g_gameOverFlag)
{
	//获取攻击状态的位图信息
```

```
BITMAP bmpHeroAction;
GetObject(g_hHeroHit[0],sizeof(bmpHeroAction),&bmpHeroAction);

if(g_HeroActionType==ACTION_TYPE_WAVEBOXING)
{
	//人物招式贴图
	SelectObject(g_bufdc,g_hHeroHit[g_HeroActionType]);
	TransparentBlt(g_hdcMem,
		g_iHeroX,g_iHeroY,
		bmpHeroAction.bmWidth/6,bmpHeroAction.bmHeight,
		g_bufdc,
		g_iHeroFileNum*bmpHeroAction.bmWidth/6,0,
		bmpHeroAction.bmWidth/6,bmpHeroAction.bmHeight,
		RGB(255,255,255));
	//绝招效果贴图
	SelectObject(g_bufdc,g_hHeroWave);
	if (g_iHeroWavesNum!=0)
	{
		for (int i=0;i<10;i++)
			if (heroWaves[i].exist)
			{
				TransparentBlt(g_hdcMem,
					heroWaves[i].x+45,heroWaves[i].y,
					bmpHeroWave.bmWidth,bmpHeroWave.bmHeight,
					g_bufdc,
					0,0,
					bmpHeroWave.bmWidth,bmpHeroWave.bmHeight,
					RGB(255,255,255));
				heroWaves[i].x+=20;
				//Hero 打出的波距离 Robot 小于 20 个像素就认为是打中
				if((g_iRobotX-heroWaves[i].x)<20)
				{
					if(g_iRobotStrength>0)
					{
						g_robotBeAttactedHeavyFlag=true;
						//造成 5-15 之间的伤害
						float damage=5+(float)(rand()%10);
						g_iRobotStrength-=damage;
					}
					else
					{
						g_iRobotStrength=0;
						g_gameOverFlag=true;
					}
```

```
                        g_RobotActionType=ACTION_TYPE_NULL;
                        g_iHeroWavesNum--;
                        heroWaves[i].exist=false;
                    }
                    //没打中到达窗口边缘自动消失
                    if (heroWaves[i].x>wndRect.right-bmpHeroWave.bmWidth)
                    {
                        g_iHeroWavesNum--;
                        heroWaves[i].exist=false;
                    }
                }
        }
}
else//普通招式贴图
{
        if(!g_gameOverFlag)
        {
            SelectObject(g_bufdc,g_hHeroHit[g_HeroActionType]);
            TransparentBlt(g_hdcMem,
                g_iHeroX,g_iHeroY,
                bmpHeroAction.bmWidth/6,bmpHeroAction.bmHeight,
                g_bufdc,
                g_iHeroFileNum*bmpHeroAction.bmWidth/6,0,
                bmpHeroAction.bmWidth/6,bmpHeroAction.bmHeight,
                RGB(255,255,255));
            //位图差距为 116
            if((g_iRobotX-g_iHeroX)<=120)
            {
                g_robotBeAttactedLightFlag=true;
                if (g_iRobotStrength>0)
                {
                    //造成 0-5 之间的伤害
                    float damage=(float)(rand()%5);
                    g_iRobotStrength-=damage;
                    Sleep(10);
                }
                else
                {
                    g_iRobotStrength=0;
                    g_gameOverFlag=true;
                }
            }
        }
}
```

```
            //g_heroActionFlag=false;
        }

            //Hero 受到轻击打
        if (!g_gameOverFlag&&g_heroBeAttactedLightFlag&&g_RobotActionType!=ACTION_TYPE_NULL)
        {
            g_heroActionFlag=false;
            SelectObject(g_bufdc,g_hHeroBeAttactedLight);
            TransparentBlt(
                g_hdcMem,
                g_iHeroX,g_iHeroY,
                bmpHero.bmWidth/6,bmpHero.bmHeight,
                g_bufdc,
                g_iHeroFileNum*bmpHero.bmWidth/6,0,
                bmpHero.bmWidth/6,bmpHero.bmHeight,
                RGB(255,255,255));
            g_heroBeAttactedLightFlag=false;
        }

        //Hero 贴图，针对其各个方向进行贴图
        if
(!g_gameOverFlag&&!g_heroActionFlag&&!g_heroBeAttactedLightFlag&&!g_heroBeAttactedHeavyFlag)
        {
            SelectObject(g_bufdc,g_hHeroDirection[g_iHeroDirection]);
            TransparentBlt(g_hdcMem,
                g_iHeroX,g_iHeroY,
                bmpHero.bmWidth/6,bmpHero.bmHeight,
                g_bufdc,
                g_iHeroFileNum*bmpHero.bmWidth/6,0,
                bmpHero.bmWidth/6,bmpHero.bmHeight,
                RGB(255,255,255));
        }

        //Robot 受到轻击打
        if (!g_gameOverFlag&&g_robotBeAttactedLightFlag/*&&g_HeroActionType!=ACTION_TYPE_NULL*/)
        {
            g_robotActionFlag=false;
            SelectObject(g_bufdc,g_hRobotBeAttactedLight);
            TransparentBlt(
                g_hdcMem,
                g_iRobotX,g_iRobotY,
                bmpRobot.bmWidth/6,bmpRobot.bmHeight,
                g_bufdc,
                g_iRobotFileNum*bmpRobot.bmWidth/6,0,
```

```
                bmpRobot.bmWidth/6,bmpRobot.bmHeight,
                RGB(255,255,255));
        //g_robotBeAttactedLightFlag=false;
}
//Robot 受到波的重击
if(g_robotBeAttactedHeavyFlag)
{
        g_robotActionFlag=false;
        SelectObject(g_bufdc,g_hRobotBeAttactedHeavy);
        TransparentBlt(g_hdcMem,
                g_iRobotX,g_iRobotY,
                g_bmpRobotHit.bmWidth/6,g_bmpRobotHit.bmHeight,
                g_bufdc,
                g_iRobotFileNum*g_bmpRobotHit.bmWidth/6,0,
                g_bmpRobotHit.bmWidth/6,g_bmpRobotHit.bmHeight,
                RGB(255,255,255));
        g_robotBeAttactedHeavyFlag=false;
}
//Sleep(500);
//g_robotActionFlag=true;
if ((!g_gameOverFlag)&&(!g_heroActionFlag)&&(g_HeroActionType==ACTION_TYPE_NULL))
{
        g_robotActionFlag=true;
        //int probability=rand()%2;
        if(1==rand()%2)
        {
                g_RobotActionType=ACTION_TYPE_KICK;
        }
        else
        {
                g_RobotActionType=ACTION_TYPE_BOXING;
        }
}
//Robot 攻击
if(!g_gameOverFlag&&g_robotActionFlag/*&&(g_heroActionFlag==false)*/)
{
        SelectObject(g_bufdc,g_hRobotHit[g_RobotActionType]);
        TransparentBlt(g_hdcMem,
                g_iRobotX,g_iRobotY,
                g_bmpRobotHit.bmWidth/6,g_bmpRobotHit.bmHeight,
                g_bufdc,
                g_iRobotFileNum*g_bmpRobotHit.bmWidth/6,0,
                g_bmpRobotHit.bmWidth/6,g_bmpRobotHit.bmHeight,
                RGB(255,255,255));
```

```
            if((g_iRobotX-g_iHeroX)<=110)
            {
                g_heroBeAttactedLightFlag=true;
                if (g_iHeroStrength>0)
                {
                    //造成 5-10 之间的伤害
                    float damage=(float)(5+rand()%5);
                    g_iHeroStrength-=damage;
                    //Sleep(10);
                }
                else
                {
                    g_iHeroStrength=0;
                    g_gameOverFlag=true;
                }
            }
        }
        //Robot 贴图
        if
(!g_gameOverFlag&&!g_robotActionFlag&&!g_robotBeAttactedLightFlag&&!g_robotBeAttactedHeavyFlag)
        {
            SelectObject(g_bufdc,g_hRobotDirection[g_iRobotDirection]);
            TransparentBlt(g_hdcMem,
                g_iRobotX,g_iRobotY,
                bmpRobot.bmWidth/6,bmpRobot.bmHeight,
                g_bufdc,
                g_iRobotFileNum*bmpRobot.bmWidth/6,0,
                bmpRobot.bmWidth/6,bmpRobot.bmHeight,
                RGB(255,255,255));
//          g_robotBeAttactedLightFlag=false;
//          g_robotBeAttactedHeavyFlag=false;
        }

        //血量位图中间的 KO logo 贴图
        SelectObject(g_bufdc,g_hLogoKO);
        StretchBlt(g_hdcMem,
            wndRect.right/2-g_bmpKO.bmWidth/2,0,
            g_bmpKO.bmWidth,g_bmpKO.bmHeight,
            g_bufdc,
            0,0,
            g_bmpKO.bmWidth,g_bmpKO.bmHeight,
            SRCCOPY);
        //贴上 Hero 血量图，根据体力值实时计算贴图位置
```

```
SelectObject(g_bufdc,g_hHeroBlood);
StretchBlt(g_hdcMem,
    (int)(wndRect.right/2*(float)(1.0-(float)(g_iHeroStrength/CHARACTER_MAX_STRENGTH))),0,

(int)((float)(g_iHeroStrength/CHARACTER_MAX_STRENGTH)*wndRect.right/2)-g_bmpKO.bmWidth/2,
g_bmpHeroBlood.bmHeight,
    g_bufdc,
    0,0,

(int)(g_bmpHeroBlood.bmWidth*(float)(g_iHeroStrength/CHARACTER_MAX_STRENGTH)),g_bmpHeroBlood.bmHeight,
    SRCCOPY);
//贴上 Robot 血量图，根据体力值实时计算贴图起始位置
SelectObject(g_bufdc,g_hRobotBlood);
StretchBlt(g_hdcMem,
    wndRect.right/2+g_bmpKO.bmWidth/2,0,

(int)(wndRect.right/2*(float)(g_iRobotStrength/CHARACTER_MAX_STRENGTH)),g_bmpRobotBlood.bmHeight,
    g_bufdc,
    0,0,
    g_bmpRobotBlood.bmWidth,g_bmpRobotBlood.bmHeight,
    SRCCOPY);

if(g_gameOverFlag)//游戏结束
{
    if (g_iHeroStrength<=0)//Hero 阵亡
    {
        //Hero 倒地
        SelectObject(g_bufdc,g_hHeroDie);
        TransparentBlt(g_hdcMem,
            g_iHeroX,g_iHeroY+10,
            g_bmpHeroHit.bmWidth/6,g_bmpHeroHit.bmHeight,
            g_bufdc,
            g_iHeroFileNum*g_bmpHeroHit.bmWidth/6,0,
            g_bmpHeroHit.bmWidth/6,g_bmpHeroHit.bmHeight,
            RGB(255,255,255));
        //Robot 站着
        SelectObject(g_bufdc,g_hRobotDirection[g_iRobotDirection]);
        TransparentBlt(g_hdcMem,
            g_iRobotX,g_iRobotY,
            bmpRobot.bmWidth/6,bmpRobot.bmHeight,
            g_bufdc,
            g_iRobotFileNum*bmpRobot.bmWidth/6,0,
```

```
                bmpRobot.bmWidth/6,bmpRobot.bmHeight,
                RGB(255,255,255));
        }
        else if(g_iRobotStrength<=0)//Robot 阵亡
        {
            //robot 倒地
            SelectObject(g_bufdc,g_hRobotDie);
            TransparentBlt(g_hdcMem,
                g_iRobotX,g_iRobotY+10,
                g_bmpRobotHit.bmWidth/6,g_bmpRobotHit.bmHeight,
                g_bufdc,
                g_iRobotFileNum*g_bmpRobotHit.bmWidth/6,0,
                g_bmpRobotHit.bmWidth/6,g_bmpRobotHit.bmHeight,
                RGB(255,255,255));
            //加载 Hero Win 的图片
            SelectObject(g_bufdc,g_hHeroWin);
            TransparentBlt(g_hdcMem,
                g_iHeroX,g_iHeroY,
                bmpHeroWin.bmWidth/6,bmpHeroWin.bmHeight,
                g_bufdc,
                g_iHeroFileNum*bmpHeroWin.bmWidth/6,0,
                bmpHeroWin.bmWidth/6,bmpHeroWin.bmHeight,
                RGB(255,255,255));
        }
        //游戏结束画面加载
        SelectObject(g_bufdc,g_hTryAgain);
        BitBlt(g_hdcMem,
            wndRect.right/4,0,
            bmpTryAgain.bmWidth/2,bmpTryAgain.bmHeight,
            g_bufdc,
            bmpTryAgain.bmWidth/2,0,SRCAND);
        BitBlt(g_hdcMem,
            wndRect.right/4,0,
            bmpTryAgain.bmWidth/2,bmpTryAgain.bmHeight,
            g_bufdc,
            0,0,SRCPAINT);
        //Hero 的攻击状态标识全部置为 false
        g_heroActionFlag=false;
        g_HeroActionType=ACTION_TYPE_NULL;

}
//最终的显示
BitBlt(g_hdc,
        0,0,
```

```
            wndRect.right,wndRect.bottom,
            g_hdcMem,
            0,0,SRCCOPY);

        //记录此次绘图时间，供下次游戏循环中判断是否已经达到画面更新操作设定的时间间隔
        g_tHeroPre = GetTickCount();
        g_iFireFrameNum++;
        g_iHeroFileNum++;
        g_iRobotFileNum++;
        if (g_iFireFrameNum==6)
        {
            g_iFireFrameNum=0;
        }
        if (g_iRobotFileNum==6)
            g_iRobotFileNum=0;

        if(g_iHeroFileNum==6)
            g_iHeroFileNum=0;
}

//游戏资源句柄释放函数
BOOL Game_CleanUp(HWND hwnd )
{
        //释放各种资源句柄
        for(int i=0;i<3;i++)
        {
            DeleteObject(g_hHeroDirection[i]);
            DeleteObject(g_hHeroHit[i]);
            DeleteObject(g_hRobotHit[i]);
            DeleteObject(g_hRobotDirection[i]);
        }
        DeleteObject(g_hTryAgain);
        DeleteObject(g_hHeroWin);
        DeleteObject(g_hHeroDie);
        DeleteObject(g_hRobotDie);
        DeleteObject(g_hFire);
        DeleteObject(g_hSnow);
        DeleteObject(g_hStart);
        DeleteObject(g_hLoading);
        DeleteObject(g_hLogoKO);
        DeleteObject(g_hHeroBlood);
        DeleteObject(g_hRobotBlood);
        DeleteObject(g_hHeroBeAttactedHeavy);
        DeleteObject(g_hHeroBeAttactedLight);
```

```
        DeleteObject(g_hRobotBeAttactedLight);
        DeleteObject(g_hRobotBeAttactedHeavy);
        DeleteObject(g_hHeroWave);
        DeleteObject(g_hRobotWave);
        DeleteObject(g_bufdc);
        DeleteDC(g_hdcMem);
        ReleaseDC(hwnd,g_hdc);
        return TRUE;
}
```